Report on Project I

Bin Fan

February 11, 2011

Abstract

We implemented a software package which can deal with simplest boundary laplacian equations. The algorithm has been implemented on a very primitive level which can only do the computation on uniform meshgrids without any accelations. In this report, we first describe our problems, the Relaxation algorithms, and our implementation details. After that, we present our solutions and the analyses of algorithm performance. Finally we did some experimental work using the algorithm.

1 Software Manual

Problem Description In this project, we are trying to attact a simple laplacian equation with relaxation methods. Our problem can be described by

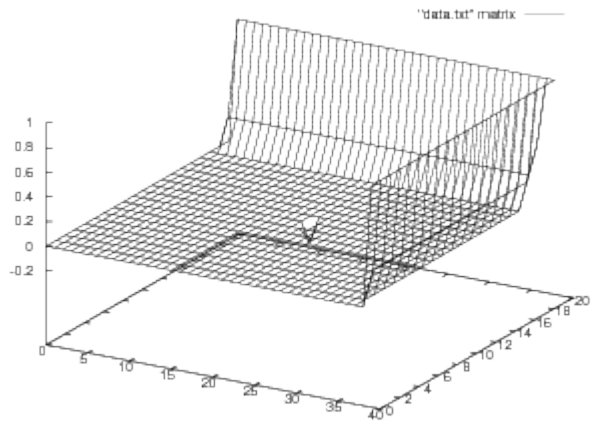
$$\Delta^2\Psi(\vec{r}) = -4\pi\rho(\vec{r})$$

Specifically in cartsian coordinates in 3 spatial dimension:

$$\frac{\partial^2}{\partial x^2}\Psi(x,y,z) + \frac{\partial^2}{\partial y^2}\Psi(x,y,z) + \frac{\partial^2}{\partial z^2}\Psi(x,y,z) = -4\pi\rho(x,y,z)$$

Also, to solve the equation, special boundary conditions are also needed. In this project, the boundary condition is assigned as figure 1 shows:

Figure 1: The initial condition of our problem



Algorithm Introduction In relaxation method, the function has been expanded in Taylor's manner:

$$f(x \pm h) = f(x) \pm hf'(x) + \frac{h^2}{2} f''(x) + \mathrm{O}\big(h^3\big)$$

Thus we have

$$f'(x) = \frac{f(x+h) - f(x-h)}{2h}$$

and

$$f(x+h) + f(x-h) = 2f(x) + h^2 f''$$

finally, we extend it to two dimension, and on the second order,

$$f(x,y) = \frac{1}{4}[f(x+h,y) + f(x-h,y) + f(x,y+h) + f(x,y-h) + (4\pi)h^2\rho(x,h)]$$

In order to solve the laplacian problem, we need to specify both the boundary condition and the source term $((4\pi)h^2\rho(x,y))$. After the two parameters have been assigned, the program will then construct a meshgrid and assign initial guess value to each node, afterward the program will recursively perform the computation until the problem converges.

Implementation Details In order to provide a better user interface, we construct three meshgrids with the same resolution. The first meshgrid, which is also the primary one, is used to do the computation. The second meshgrid will record the source term. The third one is used to store each step's recursive correction amount on each node, which is important in deciding the converging criteria.

Since the relaxation method is a recursive converging algorithm, the inital guess value must be specified before the program can be initiated. We either set all initial values to be 0 or random number between 0 and 1. User can switch between the modes by feeding different commandline argument.

During the computation of each step, the initial value and boundary condition are stored on the first meshgrid and user-define source term is on the second one. We perform the laplacian computation on the first meshgrid and the add the source term from the second meshgrid to the first one, then we copy the correction value onto the third meshgrid for the comparison with the tolerence to decide if the converging criteria has been met.

Since we are using additional meshgrid, we can easily specify source terms of each node on it without altering original program each time.

Program Usage Type ./relaxation in the directory will envoke the program, but user also need to specify algorithm, maximum iteration steps, plot or not by feeding command line arguments.

-n *[Maximum Iteration number]* : fix maximum iteration number, program will exit when loops reaches this number even if the solution has not converged yet

-p : if we want to output animation, program will remain silent defaultly

-a *[gs, jacobi, over]* : specify algorithm used

-i : if we want to set initial guess to be random numbers, otherwise the initial guess will be 0

-t *[tolerance]* : set custom tolerance, otherwise it will be 0.05%. Note that percentage, rather than absolute value is used here.

-w *[over relaxation parameter]* : set ω in relaxation method, used acompany with *-i over*, otherwise the ω is set to be 1.2

Example:

./relaxation -p -i -a jacobi -t 0.5

Perform computation with an animation output, jacobi algorithm is used and initial guess values are random numbers between 0 and 1, tolerance is

set to be 0.5%

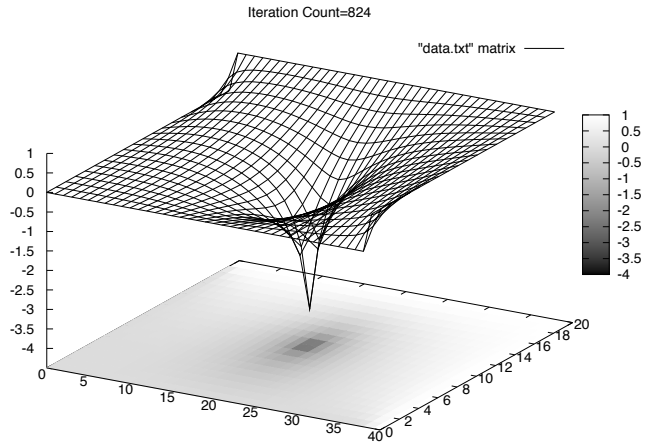
Converging Criteria As noted before, we set the third meshgrid to store each step's correction amount. The main program will check this meshgrind, compute the maximum value and average value, then decide if the two values have been both decreased to below the tolerance. Even after the tolerance is met, the program will run 100 steps further and check the criteria again to see whether the converging is stable or not.

This is a very conserved converging criteria since we want both the maximum correction amount and average one to be both decreased to below the tolerance level. I speculate sometimes there are some extremely volatile position which will fluctuate much larger than other positions. In such cases we may waste time for the program to carry on working since those flucturation may not be important to us. And the insurance 100 steps more running will bring some advantage, we have an example in the following chapter.

2 Final Result of Relaxation

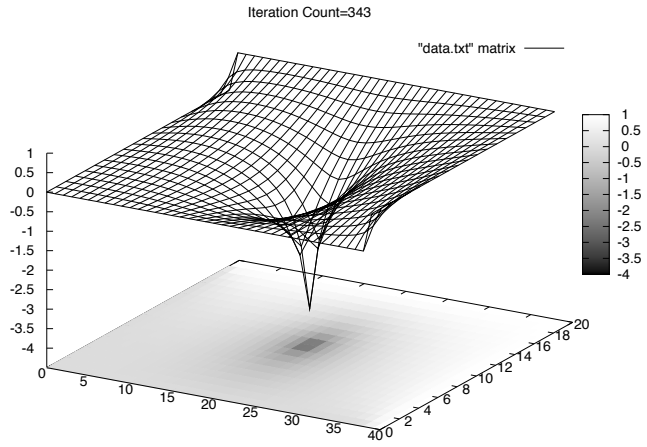
In order to compare out solutions from different algorithms, we set the tolerance level to be 0.005% and present the final solution from each algorithm below:

Figure 2: Gauss-Siedal method



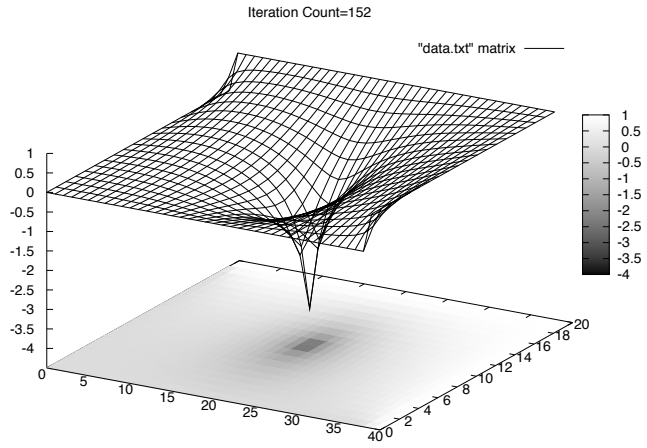
Gauss-Siedal

Figure 3: Jacobi method



Jacobi

Figure 4: Over-relaxation method with $\omega = 1.2$



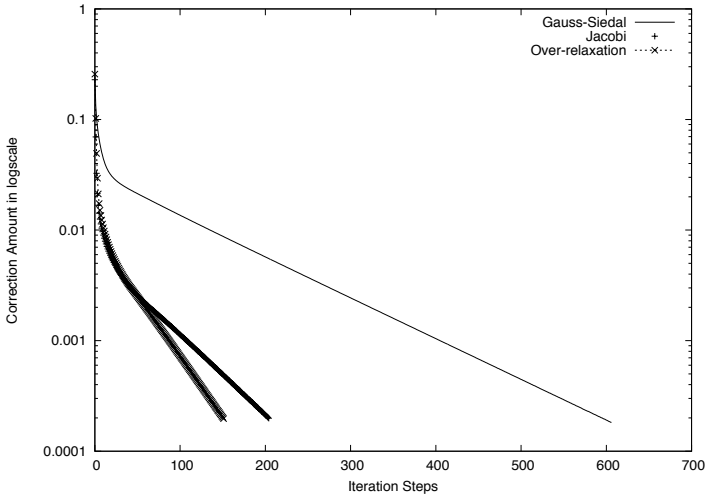
Over-relaxation It is not surprising that we cannot discern any significant difference between the final results from the three algorithms. However, the iteration steps required by different algorithms vary significantly, which also show to us the quality of the three algorithms. Surely the comparison result is Over-relaxation > Jacobi > Gauss-siedal

3 Converging Analyses

I speculate the initial guess, algorithm chosen, over-relaxation parameter and meshgrid size will affect converging speed, thus we analyze them one by one.

We first compare the converging speed of each three algorithms. The average correction amount is chosen as the mark.

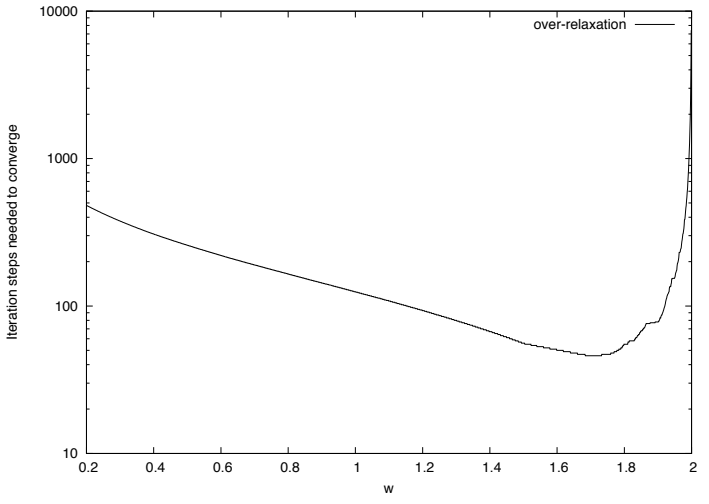
Figure 5: Converging speed of each algorithm seperately



Obviously relaxation methods with updating is a lot better than those without updating. Over-relaxation method is even better.

Then we compare the converging speed of over-relaxation method with different ω parameter

Figure 6: Converging speed vs ω in over-relaxation method

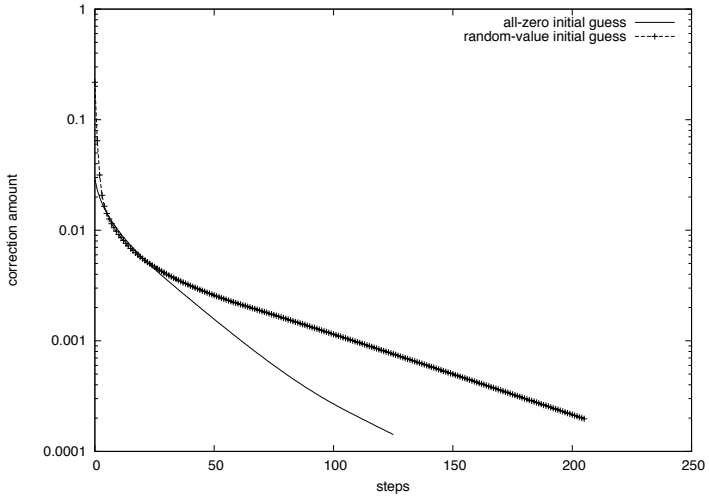


From figure 6, it is obvious that when we choose ω to be 1.7, the converging speed is fatest. As ω grows toward 2, the algorithm soon becomes

divergent.

Finally we see if the initial guess value will affect converging speed. The result below is a comparison between "all-zero" initial guess and random value guess.

Figure 7: Initial value guess vs. Converging speed



It seems that the random-value initial guess is not a good choice. This is probably because the problem we have is a smooth one. The random value initial guess actually add many fluctuations to the meshgrid which are both useless and would require more steps to smooth them.

4 Algorithm Analyses and Experiments

Let us look at the Gauss-seidel and jacobi methods in matrix terms, suppose we are solving a problem of

$$Ax = b$$

and $A = L + D + U$, in which L is lower matrix, D is diagnol part of matrix and U is upper half of matrix. Then at step n, the Jacobi method can be writen as

$$Dx^{(r)} = -(L + U)x^{(r-1)} + b$$

in which D is simply the identity matrix. However, the Gauss-seidel one is

$$(D + L)x^{(r)} = -Ux^{(r-1)} + b$$

Thus if we rewrite them as

$$x^{(r)} = Tx^{(r-1)}$$

T is the operation matrix on each step. The converging requirement for T is that the largest eigenvalue of T is smaller than 1.

For over-relaxation method with updating, we can write:

$$x^{(r)} = x^{(r-1)} - \omega(L+D)^{-1}[(L+D+U)x^{r-1} - b]$$

for over-relaxation method without updating,

$$x^{r} = [1 - \omega - \omega D^{-1}(L+U)]x^{r-1} + bD^{-1}$$

and now we see it is very dangerous to perform over-relaxation on non-updating jacobi method. The T matrix $1-\omega-\omega D^{-1}(L+U)$ can be compared to the T matrix for jacobi method $D^{-1}(L+U)$. We already know the jacobi method is extremely slow and thus its eigenvalue spectra must be close to 1, if at the same time we put that matrix to $1 - \omega - \omega D^{-1}(L+U)$, it is very possible that the spectra might exceed 1.